An Application of Genetic Programming to Electronic Design Automation: from Frequency Specifications to VHDL Code

Roberto Rossi[1], Valentino Liberali[2], and Andrea G. B. Tettamanzi[2]

[1] Università degli Studi di Pavia
Dipartimento di Elettronica
Via Ferrata 1, 27100 Pavia, Italy
roberto.rossi@ele.unipv.it
[2] Università degli Studi di Milano
Dipartimento di Tecnologie dell'Informazione
Via Bramante 65, 26013 Crema, Italy
liberali@dti.unimi.it, tettaman@dsi.unimi.it

Abstract. An evolutionary algorithm is used to design a digital filter with reduced power consumption. The proposed design approach combines genetic optimisation and simulation methodology, to evaluate a multi-objective fitness function which includes both the suitability of the filter transfer function and the transition activity of digital blocks. A client-server computer program allows the user to exploit several CPU's with parallel evolving populations, aiming at the same target specifications but with different fitness functions. Examples on design cases are presented and discussed.

1 Introduction

Electronics design automation must cope with technological trend in silicon integration. Nowadays, the possibility of integrating millions of transistors onto a single silicon chip is demanding for new CAD tools, to bridge the gap between technology capabilities and designer productivity.

Therefore, new branches in computer aided design are expected to emerge in the next future: among them, evolutionary algorithms seem to be very promising, due to their capability to provide solutions to hard design problems. Built on the key concept of Darwinian evolution in biology [1], evolutionary algorithms are a broad class of optimisation methods [2]. In the past years, they have been successfully applied to physical design (partitioning, placement and routing). Now bio-inspired electronic design methods are being considered in a variety of circumstances [3, 4]. Evolutionary algorithms for circuit synthesis are a powerful technique, which could provide innovative solutions to several design problems, also when classical decomposition methods may fail [5].

This paper presents an evolutionary approach to the design of digital filters. In a previous work, it was demonstrated that genetic programming can be used

to design digital filters starting from mask specifications, and obtaining a synthesizable VHDL code [6]. Now, practical aspects are being considered, aiming at a more efficient design tool, suitable for use in industrial environment.

2 Motivation and Problem Statement

Microintegrated technology enables the development of deep submicron CMOS circuits for digital signal processing (DSP) in a broad variety of applications. However, the ever and ever increasing integration scale makes designers to face new problems and requires new design methodologies and tools. Today's major issues are:

1. the increasing gap between technology capability and designer productivity. Such a scenario is demanding for new design methodologies and innovative CAD tools, as in previous stages of the "design crisis" [7]. Indeed, a remarkable effort is being spent, aiming at the automatic synthesis of electronic design, starting from functional specifications rather than from behavioural description;
2. the increasing power consumption. Circuits with small transistors can be densely packed into a single chip, and size reduction allows frequency to be increased. Consequently, the dynamic power P due to the logic transitions of circuit nodes also increases. It can be expressed as:

$$P = \frac{1}{2} C V^2 f \alpha \tag{1}$$

   where C is the total capacitance, V is the power supply voltage, f is the clock frequency, and α is the transition activity, i.e. the average number of logic transitions in a clock period [8, 9];
3. product testability. The increasing complexity of integrated systems leads to a huge number of internal nodes which cannot be directly controlled and/or observed. Since manufacturability requires to test every chip to determine whether it is good or bad, it is apparent that more powerful design-for-test techniques are needed.

This paper addresses issues 1 and 2, proposing an approach for the automatic generation of the digital circuits having a reduced transition activity of digital gates. Such a task is not a straightforward one, because digital activity is a non-linear function of input patterns.

3 Genetic Representation of Digital Filters

A description of digital filters can derived from their frequency response in the z-domain [10]. The frequency response of a finite impulse response (FIR) digital filter is:

$$H(z) = \sum_{k=0}^{N-1} h(k) z^{-k} \tag{2}$$

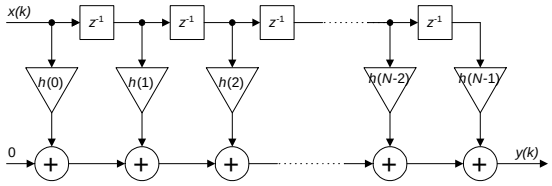

Fig. 1. Canonical direct form of a FIR filter

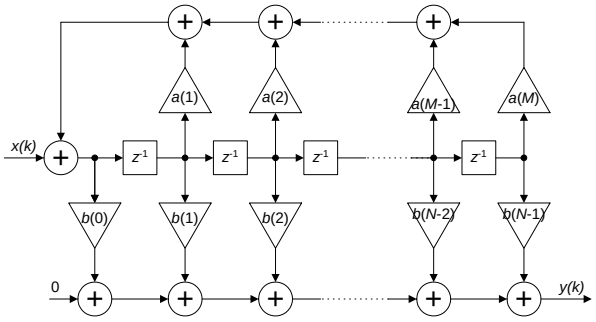

Fig. 2. Canonical direct form of an IIR filter

and the canonical direct form of the filter shown in Fig. 1.

An infinite impulse response (IIR) digital filter can be represented with the canonical form shown in Fig. 2, and its frequency response is:

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{k=0}^{N-1} b(k) z^{-k}}{1 - \sum_{j=1}^{M} a(j) z^{-j}} \tag{3}$$

To save area and to reduce power consumption, generic multiplier blocks are often replaced with shifters and adders [11]. As an example, multiplication by 13 can be implemented with two shifts and two additions, as illustrated in Fig. 3, where the block "$<< n$" means "left shift by n positions".

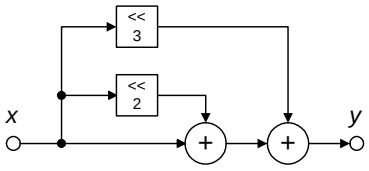

Fig. 3. Multiplication by 13 implemented with shifters and adders

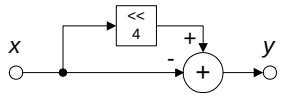

Fig. 4. Multiplication by 15 implemented with one shifter and one subtractor

The canonical signed digit (CSD) representation [12] assigns a separate sign to each digit: 0, 1 and $\bar{1}$ (= –1). Its goal is to minimise the number of non-zero digits: by encoding the filter coefficients with CSD, the filter ouput can be computed using a reduced amount of hardware, since multiplications by zero are simply not implemented. As an example, consider the multiplication by 15: since $15 = 2^3 + 2^2 + 2^1 + 2^0 = (001111)_2$, this operation in binary arithmetics would require three shifts and three additions; while using CSD we can write $15 = 2^4 - 2^0 = (0100\bar{0}\bar{1})_2$, and we implement the same operation using only one shifter and one subtractor (Fig. 4).

Starting from these considerations, a digital filter can be described using a very small number of elementary operations. The primitives selected for digital filters are listed in Table 1. Each elementary operation is encoded by its own code (one character) and by two integer numbers, which represent the relative offset (calculated from the current position) of the two operands. When all the offsets are positive (i.e. each block can receive data from previous blocks only), no feedback loop occurs and the resulting structure is a FIR filter. If the relative offsets can be either positive or negative, the resulting filter may have either a FIR or an IIR response, depending on the presence of llops in the signal flow. All primitives include a delay z^{-1}, to avoid possible problems due to timing violations during the synthesis process.

The primitive operators requiring an adder block are more expensive, both in terms of area and of power dissipation. For our purposes, we considered only power consumption: according to gate level simulations, we assigned a relative weight factor to sum, difference and complement.

Table 1. Primitives of the genetic algorithm

Name	Code	Op 1	Op 2	Description
Input	`I`	not used	not used	Copy input: $y_i = x$
Delay	`D`	n_1	not used	Store value: $y_i = y_{i-n_1} z^{-1}$
Left shift	`L`	n_1	p	Multiply by 2^p: $y_i = 2^p y_{i-n_1} z^{-1}$
Right shift	`R`	n_1	p	Divide by 2^p: $y_i = 2^{-p} y_{i-n_1} z^{-1}$
Adder	`A`	n_1	n_2	Sum: $y_i = (y_{i-n_1} + y_{i-n_2}) z^{-1}$
Subtractor	`S`	n_1	n_2	Difference: $y_i = (y_{i-n_1} - y_{i-n_2}) z^{-1}$
Complement	`C`	n_1	not used	Multiply by -1: $y_i = -y_{i-n_1} z^{-1}$

As an example, the following sequence is made of 6 primitives (6 genes):

```
(I 0 2) (D 1 3) (L 2 2) (A 2 1) (D 1 0) (S 1 5)
```

and it is interpreted as follows:

$$\begin{aligned} y_0 &= x \\ y_1 &= y_0 z^{-1} \\ y_2 &= 2^2 y_0 z^{-1} \\ y_3 &= (y_1 + y_2) z^{-1} \\ y_4 &= y_3 z^{-1} \\ y = y_5 &= (y_4 - y_0) z^{-1} \end{aligned} \tag{4}$$

The last value is the output of the filter. By merging the equations (4), we obtain the transfer function:

$$H(z) = \frac{y}{x} = 5z^{-4} - z^{-1} \tag{5}$$

We note that such representation of a filter has the same essence as a program in a simple imperative programming language, and we can apply genetic programming [13] or, more precisely, Cartesian genetic programming [14] to the task of designing digital filters. Fine granularity of primitives allows a simple genetic encoding, and allows the evolutionary algorithm to perform a better search within the design space.

Efficiency of the designed structure (in term of both area and power consumption) is an important target: it should be optimised under the specification constraints. Evolutionary design have been proven more efficient than structured conventional approaches [15].

It is worth remarking that the proposed genetic representation is different from the evolutionary design of filter coefficients, proposed in previous papers [16, 17]. Indeed, the representation of a filter as a sequence of elementary operations is closer to the low-level filter structure, and gives us the opportunity of evaluating directly the power consumption through the switching activity of digital nodes.

Moreover, the proposed design approach is fully compliant with the well-consolidated digital design methodology: the genetic algorithm produced the synthesizable VHDL code, while all the design tools actually used in electronic design flow are still used to translate the design in structural and in physical domains.

4 The Evolutionary Algorithm

In order to make filters evolve, a variable population size island distributed evolutionary algorithm has been implemented. Details on the genetic operators can be found in [6].

It is well known that a genetic algorithm may become very inefficient when the genome size increases: this phenomenon, known as “bloat”, must be limited as much as possible [18]. For this purpose, the algorithm uses an intelligent mutation operator, which removes useless portions of the genome.

There is very strong evidence that neutral mutations should occur, as they improve the evolvability [19]. Although, this subject is still undere debate. Our results appear to contribute additional evidence to this view. Since neutral mutations conflict with bloat limitation, a suitable trade-off has to be found. Moreover, to avoid tuning of parameters for different specifications, crossover, mutation and selection parameters are dynamically adapted [20].

4.1 Fitness Function

The evolutionary algorithm must find a design that satisfies user specifications and minimises transition activity of digital gates. These two objectives, however, are hierarchical: specification fulfilment is mandatory, while reduction of digital transitions is just an additional quality, which is meaningful only when the circuit is fully compliant with requests.

Therefore, a two-step fitness function has been devised. First of all, we consider filter specifications (given as a frequency mask): the frequency range is sampled at N equally spaced frequencies ω_i, and the filter response $H(\omega_i)$ is calculated for every ω_i in the pass-band and in the stop-band. The partial error ϵ_i is set to zero, if $|H(\omega_i)|$ lies within the specifications; otherwise it is proportional to the overshoot or undershoot with respect to the given tolerance. The total error E_{tot} is simply the sum of all partial errors: it is zero when the frequency response of the filter lies within the mask; otherwise, it has a non-zero value. Mask fitness is defined as:

$$f_M = \frac{1}{1 + E_{\text{tot}}} \tag{6}$$

It measures the extent to which the filter complies with frequency specifications and $f_M = 1$ when specifications are completely met.

The second step is the evaluation of the activity fitness, defined as:

$$f_A = 1 + \frac{a}{N_T} \tag{7}$$

where N_T is the number of weighted digital transitions per input sample and a is a constant; its contribution is higher for filters with low transition activity, which is responsible for power consumption.

Finally, the overall fitness is defined as:

$$f = \begin{cases} f_M & \text{if } E_{\text{tot}} > 0 \\ f_M + f_A & \text{otherwise} \end{cases} \tag{8}$$

Such a definition guarantees that circuits complying with specifications have a larger fitness than circuits with a response outside the frequency mask.

4.2 Selection

Different selection strategies have been implemented:

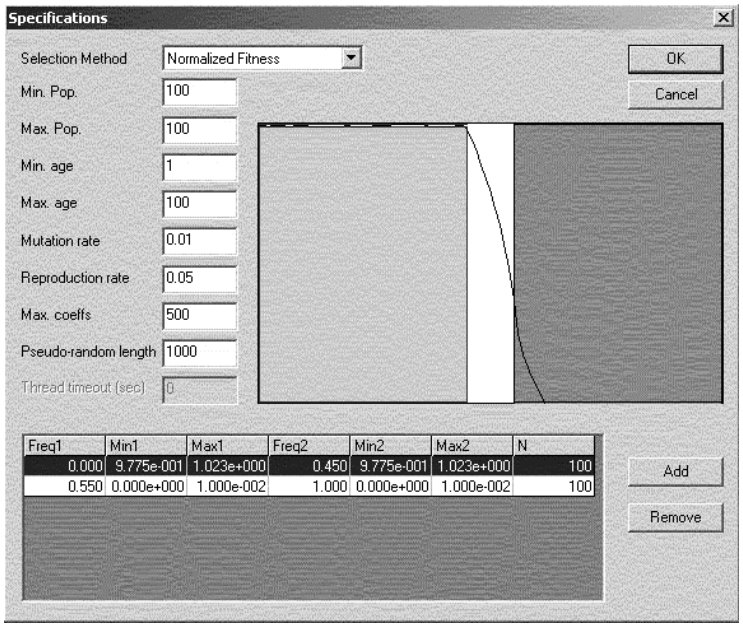

Fig. 5. Client graphical interface with frequency mask

1. an "aging" rule, which assigns a life expectation to every individual at its birth, according to its ranking: life expectation linearly decreases from 100 to 1 generation as the ranking goes from best to worst;
2. fitness-proportionate selection, whereby each individual has a survival probabaility equal to its normalised fitness;
3. linear ranking selection with elitism, whereby the best two individuals survive with probability $P_s = 1$, and the individual of rank i has a probability $P_s = 1 - \frac{i}{2\bar{n}}$ of surviving, where $\bar{n}$ is a parameter of the algorithm affecting the average population size;
4. selection based on normalised fitness and ranking, whereby each individual has a survival probability $P_s = \sqrt{f_i' r_i'}$ where f_i' and r_i' are the normalised fitness and rank;
5. histogram-based selection, whereby the selection aims at a uniform distribution of the fitness.

Experimental evidence shows that each selection strategy performs better in a particular stage of the evolution. For instance, we have seen that the "aging" selection favours a faster evolution at early stages, while normalised fitness and ranking selection give better results later on. However, a detailed discussion on selection strategies is beyond the scope of this paper. In the examples presented below, all strategies have been used in parallel.

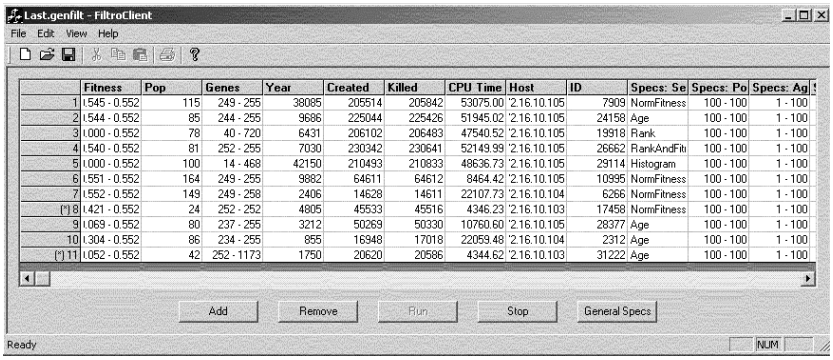

Fig. 6. Client interface with thread list

5 Client-Server Implementation

The algorithm has been implemented with a client-server approach. Communication between the two parts occurs through a TCP/IP socket. The server part may run on several computers, and accepts connections from a client. The client has a graphical user interface, that allows us to input the design parameters. As an example, Fig. 5 illustrates the mask specifications of the filter (i.e., the minimum and maximum attenuation as a function of the normalised frequency).

Using the graphical client interface, a connection to the servers is established and simulated evolutions are run in parallel. Each server starts a new thread for every request from the client. Different threads may have different parameters. The client window is shown in Fig. 6.

Individuals are randomly sent from one thread to another thread, to maximise the success probability of the algorithm. Moreover, when an individual is copied to a different thread, its neutral genes are removed from the genome. This intelligent mutation greatly reduces the probability of bloat.

6 Design Examples

The proposed design method has been applied to the design of two digital filters to be used as decimation stages in a $\Sigma\Delta$ analog-to-digital converter. Filter specifications are listed in Table 2.

Both filters were previously designed using the `remez` function available in the Matlab Signal Processing Toolbox [21], and by approximating ideal coefficients with finite precision numbers.

Using the evolutionary algorithm, we were able to obtain designs with better characteristics in terms of both area and power consumption.

For design #1, the genome of the evolved filter has 46 primitives and the resulting frequency response is shown in Fig. 7. It is apparent that the filter meets all design specifications.

Table 2. Filter specifications

	filter #1	filter #2
clock frequency	64 MHz	64 MHz
data input rate	16 MHz	8 MHz
data output rate	8 MHz	4 MHz
normalised pass-band	0 ... 0.25	0 ... 0.45
maximum pass-band ripple	0.2 dB	0.2 dB
normalised stop-band	0.75 ... 1	0.55 ... 1
minimum stop-band attenuation	60 dB	40 dB
normalised "don't care" band	0.25 ... 0.75	0.45 ... 0.55

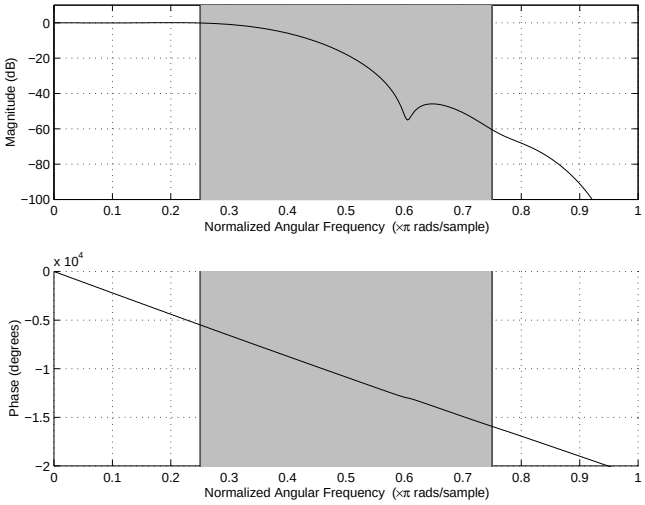

Fig. 7. Frequency response of the evolved filter #1; the grey area represents the "don't care" band

Fig. 8 illustrates the VHDL code generated by the algorithm. The RTL (register-transfer logic) architecture [22] is translated by Synopsys into a circuit contining 10,000 equivalent logic gates. The conventional filter implementation required about 40,000 gates to meet the same specifications. By comparing these two figures, we can conclude that the effort of the evolutionary algorithm towards optimisation of transition activity also led to a dramatic reduction of the required hardware, thus reducing the silicon area (and hence the cost) by a factor of 4. From simulation with a pseudorandom input pattern, the evolved filter has a power consumption of 14 mW, while the previously designed filter dissipates as much as 40 mW.

```
entity filter is
port(
  y:out std_logic_vector(29 downto 0);
  clk:in std_logic;
  rst:in std_logic;
  x:in std_logic_vector(15 downto 0)
);
end entity;

architecture RTL of filter is
--filter signals
signal y0: std_logic_vector(16 downto 0);
signal y1: std_logic_vector(16 downto 0);
...
signal y45: std_logic_vector(29 downto 0);

begin
y <= y45;
gene0: adder generic map(Bits_x1=>x'length,
   Bits_x2=>x'length, Bits_y=>y0'length)
   port map(x1=>x, x2=>x, y=>y0, clk=>clk, rst=>rst);
gene1: change_s generic map(Bits=>y2'length)
   port map(x=>x, y=>y1, clk=>clk, rst=>rst);
...
gene45: adder generic map(Bits_x1=>y43'length,
   Bits_x2=>y42'length, Bits_y=>y45'length)
   port map(x1=>y43, x2=>y42, y=>y45, clk=>clk, rst=>rst);
end RTL;
```

Fig. 8. VHDL synthesisable code generated by the algorithm

The filter #2 is more demanding, due to its steeper transition band. Its evolutionary design required several days of CPU time on five computers running in parallel. Fig. 9 shows its frequency response.

Finally, Table 3 compares the characteristics of evolutionary designs with conventional ones.

7 Conclusion

This paper has described an evolutionary approach to the design of digital filters. Genetic encoding of filter primitives has a fine granularity which is exploited by the evolutionary algorithm during its random search. The encoding is also a straightforward representation of the time-domain response of the filter, thus allowing a direct evaluation of cost and transition activity of digital blocks. A fitness function has been devised to allow multi-objective evolution. The "best" result produced by the algorithm is automatically translated into VHDL code,

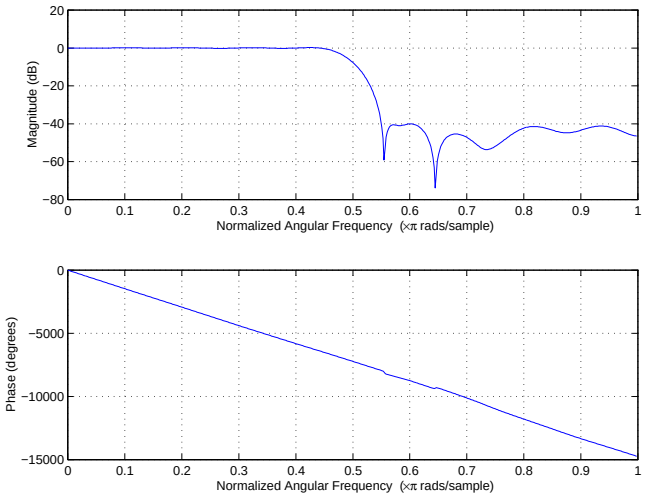

Fig. 9. Frequency response of the evolved filter #2

Table 3. Characteristics of the filters (after synthesis)

	filter #1		filter #2	
	evolutionary	conventional	evolutionary	conventional
No. of coefficients	17	13	104	101
No. of primitives	32	60	220	309
No. of logic gates	10,000	40,000	69,000	105,000
Power dissipation	14 mW	40 mW	37 mW	59 mW

which can be synthesised into a circuit without any additional operation, according to the standard digital design methodology. Client-server implementation of the evolutionary algorithm allows to run multiple threads in parallel.

The results obtained with the simulated evolution show that minimisation of transition activity leads to a dramatic reduction of the hardware with respect to the conventional design methodology, while maintaining the same filter performance.

References

1. Darwin, C.: On the Origin of Species by Means of Natural Selection. John Murray, London, UK (1859)
2. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, Oxford, UK (1996)
3. Drechsler, R.: Evolutionary Algorithms for VLSI CAD. Kluwer Academic Publishers, Dordrecht, The Netherlands (1998)

4. Sipper, M., Mange, D., Sanchez, E.: Quo vadis evolvable hardware? Communications of the ACM **42** (1999) 50–56
5. Thompson, A., Layzell, P.: Analysis of unconventional evolved electronics. Communications of the ACM **42** (1999) 71–79
6. Erba, M., Rossi, R., Liberali, V., Tettamanzi, A.G.B.: An evolutionary approach to automatic generation of VHDL code for low-power digital filters. In: Genetic Programming – Proc. 4th European Conference (EuroGP2001), Como, Italy (2001) 36–50
7. Wakabayashi, K., Okamoto, T.: C-based SoC design flow and EDA tools: An ASIC and system vendor perspective. IEEE Trans. Computer-Aided Design of Integr. Circ. and Syst. **19** (2000) 1507–1522
8. Tsui, C.Y., Monteiro, J., Pedram, M., Devadas, S., Despain, A.M., Lin, B.: Power estimation methods for sequential logic circuits. IEEE Trans. VLSI Systems **3** (1995) 404–416
9. Pedram, M.: Power minimization in IC design: Principles and applications. ACM Trans. on Design Automation of Electronic Systems **1** (1996) 3–56
10. Jackson, L.B.: Signals, Systems, and Transforms. Addison-Wesley, Reading, MA, USA (1991)
11. Zhao, Q., Tadokoro, Y.: A simple design of FIR filters with power-of-two coefficients. IEEE Trans. Circ. and Syst. **35** (1988) 556–570
12. Pirsch, P.: Architectures for Digital Signal Processing. John Wiley & Sons, Chichester, UK (1998)
13. Koza, J.R.: Genetic Programming: on the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge, MA, USA (1993)
14. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Genetic Programming – Proc. 3rd European Conference (EuroGP 2000), Edinburgh, UK (2000) 121–132
15. Miller, J.F., Job, D., Vassilev, V.K.: Principles in the evolutionary design of digital circuits – Part I. Genetic Programming and Evolvable Machines **1** (2000) 7–35
16. Harris, S.P., Ifeachor, E.C.: Automatic design of frequency sampling filters by hybrid genetic algorithm techniques. IEEE Trans. Signal Processing **46** (1998) 3304–3314
17. Lee, A., Ahmadi, M., Jullien, G.A., Lashkari, R.S., Miller, W.C.: Design of 1-D FIR filters with genetic algorithm. In: Proc. IEEE Int. Symp. on Circ. and Syst. Volume III., Orlando, FL, USA (1999) 295–298
18. McPhee, N.F., Poli, R.: A schema theory analysis of the evolution of size in genetic programming with linear representation. In: Genetic Programming – Proc. 4th European Conference (EuroGP 2001), Como, Italy (2001) 108–125
19. Yu, T., Miller, J.: Neutrality and evolvability of boolean function landscape. In: Genetic Programming – Proc. 4th European Conference (EuroGP 2001), Como, Italy (2001) 204–217
20. Niehaus, J., Banzhaf, W.: Adaption of operator probabilities in genetic programming. In: Genetic Programming – Proc. 4th European Conference (EuroGP 2001), Como, Italy (2001) 325–336
21. The Mathworks, Inc.: Signal Processing Toolbox, Natick, MA, USA. (1983)
22. Weste, N.H.E., Eshraghian, K.: Principles of CMOS VLSI Design: a System Perspective (2nd edition). Addison-Wesley, Reading, MA, USA (1993)